HWP7 – Protokoll

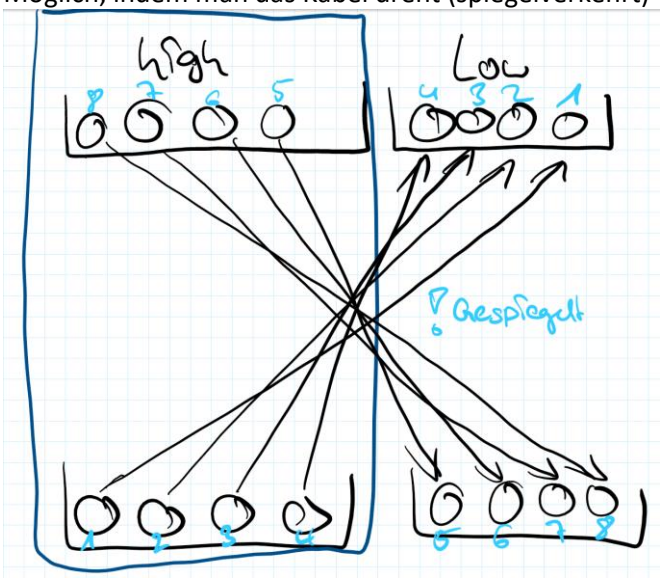
Max [redacted] INB-2 [redacted]
T [redacted] P [redacted] INB-2 [redacted]

02.02.2024

1. Vorüberlegungen:

1.1 Vollduplex-Datenübertragung:

- 8Bit Leitung:
  Im Programm: schicken auf oberen 4 Bits, lesen auf den unteren 4 Bits
  DDRA = 0b11110000
  Möglich, indem man das Kabel dreht (spiegelverkehrt)



  Spiegelung beachten!
  Dafür reverse Funktion, welche Spiegelung umkehrt

```
char reverse(char b){
    b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
    b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
    return b;
}
```

  ! Weil wir nur auf den oberen 4 Bits schicken, 8BIT Char in 2x4Bit aufteilen und das untere Bit hochshiften (<<4)

1.2 Aufteilung der Chars in 4Bit Pakete:

- Start-Bitmuster : 0000
- End-Bitmuster : 1111
- ESC – Bitmuster (8Bit, um Wahrscheinlichkeit zu verringern, dass es als normale Nachricht auftritt)
  - ESC1 : 0001
  - ESC2 : 1011

Einschieben von ESC-Bits, wenn die gleichen 4 Bits 2-mal hintereinander geschickt werden sollten.
(0000 1010 0001 1011 1010 1111)

- Falls wirklich das ESC-Zeichen geschickt werden soll, wird dies 2-mal hintereinander geschickt
  (0000 0001 1011 0001 1011 1111)

- Startsignal (senden) : 11000000
- Startsignal (empfangen) : 00000011 (wegen Kabeldrehen)
- Endisignal Bitkombination : (Kabeldrehen aufpassen)

```
std::vector <uint8_t> schickeEndsignal = {
    0b00000000, 0b10000000, 0b01000000, 0b11000000, 0b10100000, 0b01100000,
    0b11100000, 0b01110000, 0b11110000};
std::vector <uint8_t> empfangeEndsignal = {
    0b00000000, 0b00000010, 0b00000010, 0b00000011, 0b00000101, 0b00000110,
    0b00000111, 0b00001110, 0b00001111};
```

2. Programmablaufs-Plan:

2.1 Programmstart / Synchronisation:

- Lese zu schickende Eingabe und speichere alle Chars in einem Vector
- Schicke Startsignal an anderen PC2 und warte in einer While Schleife auf das Startsignal von PC2
- Wenn Startsignal erhalten, starte (schicken & lesen)^{X} (lesen)^{Y}.

2.2 Schicken & Lesen:

- Lese Char aus Vector ein, Teile ihn in 4Bit Pakete (Siehe: 1.2 Aufteilung der Chars in 4Bit Pakete) und speichere sie in einen PaketSendenVector
- Schicke Paket aus PaketSendenVector und lese den PINA
  - lese PINA Paket, speichere wenn es != das vorherig gelesene Paket ist
  - bei Erkennung des Endbit, anlysiere vorherig empfangene Pakete und ermittle daraus die 2x4Bit, aus welchem das Char besteht
    Char zusammensetzen und ausgeben

```
getriggert wenn: 1111 gelesen wurde
                0       1    2    3    4    5       6       7
    1.0 Fall:   0000    xxxx yyyy 1111

    2.0 Fall:   0000    ESC1 ESC2 ESC1 ESC2 1111        CASE 6
    2.1 Fall:   0000    ESC1 ESC2 0000 xxxx 1111        CASE 6
    2.2 Fall:   0000    xxxx ESC1 ESC2 xxxx 1111        CASE 6
    2.3 Fall:   0000    xxxx 1111 ESC1 ESC2 1111        CASE 6


    4.0 Fall:   0000    1111 ESC1 ESC2 1111 ESC1 ESC2   1111
    4.1 Fall:   0000    ESC1 ESC2 0000 ESC1 ESC2 0000   1111
```

- While(schreiben){schreiben(); lesen();)
  Sobald eigenes Endsignal geschickt, wird die Prüfsumme berechnet und hinterhergeschickt
  While (nicht Endsignal gelesen){ weiterlesen(); }

Sobald Endsignal gelesen, beinhaltet die nächste Paketkombination 000 … … 1111 die Prüfsumme für den von dem anderen PC eingelesenen Text

- sobald ein PC seine Nachricht zuende geschrieben hat, schickt dieser das Endsignal (siehe 1.2)
- und schickt danach seine Prüfsumme von seiner eingelesen Eingabe (1.3)

2.3 Prüfsumme:

- PC1: eingelesene Eingabe in Vector speichern, aus diesem CRC8 Prüfsumme bilden
- PC1: von PC2 empfangene Chars werden in einen empfangeneCharsVector gespeichert
- PC1: sobald das Endsignal Empfangen wurde, wird auf die von PC2 danach gesendete Prüfsumme gewartet und aus dem Vector empfangeneCharsVector die eigene Prüfsumme gebildet und mit der erhaltenen verglichen

Fall1: Prüfsummen identisch:
Alles okay
Fall2: Sende Paketkombination, auf welche der andere PC reagiert, in dem er die eingelesene Eingabe nocheinmal von vorne schickt.

3. Code:

```
#include <iostream>
#include <string>
#include <cstdint>
#include <bitset>
#include <b15f/b15f.h>
#include <thread>
#include <chrono>
#include <array>
#include <vector>
#include <iterator>
#include <zlib.h>


using namespace std;

int dateiAuslesen(std::string dateiname);
int uebertragungUndLeseStart();
void lesen(uint8_t gelesenesPaket);
void gelesenArrayAbfragen();
void charInPaketeAufteilen(uint8_t erstesPaket, uint8_t zweitesPaket);
void lesePaketeZusammensetzen(uint8_t erstesPaket, uint8_t zweitesPaket);
void resetLeseVariablen();
char reverse(char b);
int eingabeLesen();
uint8_t berechneCRC8(const std::vector<uint8_t>& data);
//18:04
```

```
B15F& drv = B15F::getInstance();
uint8_t startBit = 0b0000 << 4;
uint8_t endBit = 0b1111 << 4;
//EscBit 0001 1011
uint8_t sendEscBit1 = 0b00010000;
uint8_t sendEscBit2 = 0b10110000;
uint8_t readEscBit1 = 0b00001000;
uint8_t readEscBit2 = 0b00001101;

char gelesenesZeichen;
std::vector<uint8_t> sendenPakete;

//Einzulesene Datei:
std::vector<char> inputText;
//Startsignal zum senden & empfangen
char eingegebenesChar;
uint8_t schickenStartsignal = 0b11000000;    // ä (zumindest der start davon)
uint8_t empfangenStartsignal = 0b00000011;   //gedrehtes ä
std::vector <uint8_t> schickeEndsignal = {
    0b00000000, //Start 1
    0b10000000, // 2
    0b01000000, // 3
    0b11000000, // 4
    0b10100000, // 5
    0b01100000, // 6
    0b11100000, // 7
    0b01110000, // 8
    0b11110000, // Ende 9
};
std::vector <uint8_t> empfangeEndsignal = {
    0b00000000, //Start 1
    0b00000001, // 2
    0b00000010, // 3
    0b00000011, // 4
    0b00000101, // 5
    0b00000110, // 6
    0b00000111, // 7
    0b00001110, // 8
    0b00001111, // Ende 9
};

//Lesevariablen
std::vector<uint8_t> gelesenePakete1;
uint8_t gelesenesErstesPaket; //Erstes herausgefundenes Paket des empfangenen
Chars
uint8_t gelesenesZweitesPaket; //Zweites herausgefundenes Paket des
empfangenen Chars

std::vector<uint8_t> alleGesendetenChars;
uint8_t eigeneDateiCRC8;
```

```
std::vector<uint8_t> alleEmpfangeneChars;
uint8_t empfangeneDateiEigenesCRC8;
uint8_t empfangeneDateiEmpfangenesCRC8;


std::vector<char> nachSchreibenLesenBisEnde;
bool weiterlesen = true;
bool prüfsummeNext = false;


std::chrono::milliseconds dauer(10); // Zum Beispiel 10 Millisekunden


int main(){
    drv.setRegister(&PORTA, 0b00000000 );
    drv.setRegister(&DDRA, 0xF0 );
    drv.setRegister(&PORTA, 0b00000000 );

    drv.setRegister(&PORTA, schickenStartsignal );
    while((drv.getRegister(&PINA) & 0x0F) != empfangenStartsignal){

    }
    std::this_thread::sleep_for(dauer);

    eingabeLesen();

    drv.setRegister(&PORTA, (schickenStartsignal & 0xF0));

    while((drv.getRegister(&PINA) & 0x0F) != empfangenStartsignal){
        //solange warten bis Startsignal empfangen wurde
    }

    uebertragungUndLeseStart();

    return 0;
}

int eingabeLesen(){

    //STDIN lesen
    char aktuellesInputChar;

    while(std::cin.get(aktuellesInputChar)){

        inputText.push_back(aktuellesInputChar);
    }

    return 0;
}
```

```
int uebertragungUndLeseStart(){

    for(int i = 0; i < inputText.size(); i++){

        alleGesendetenChars.push_back(inputText.at(i));

        uint8_t erstesPaket = reverse((inputText.at(i) & 0xF0));
        uint8_t zweitesPaket = reverse((inputText.at(i) & 0x0F)) << 4;

        charInPaketeAufteilen(erstesPaket, zweitesPaket);
        lesen( (drv.getRegister(&PINA) & 0x0F) );
        for(int j=0; j<sendenPakete.size(); j++){
            std::this_thread::sleep_for(dauer);
            drv.setRegister(&PORTA, (sendenPakete.at(j) & 0xF0) );
            lesen( (drv.getRegister(&PINA) & 0x0F) );
            std::this_thread::sleep_for(dauer);
            lesen( (drv.getRegister(&PINA) & 0x0F) );
        }
    }
    for (int i=0; i < schickeEndsignal.size(); i++){
        drv.setRegister(&PORTA, (schickeEndsignal.at(i) & 0xF0) );
        lesen( (drv.getRegister(&PINA) & 0x0F) );
        std::this_thread::sleep_for(dauer);
    }
    eigeneDateiCRC8 = berechneCRC8(alleGesendetenChars);

    charInPaketeAufteilen((eigeneDateiCRC8 & 0xF0),((eigeneDateiCRC8 &
0x0F)<<4));
    lesen( (drv.getRegister(&PINA) & 0x0F) );
    for(int i=0; i < sendenPakete.size(); i++){
        std::this_thread::sleep_for(dauer);
        drv.setRegister(&PORTA, (sendenPakete.at(i) & 0xF0) );
        lesen( (drv.getRegister(&PINA) & 0x0F) );
        std::this_thread::sleep_for(dauer);
        lesen( (drv.getRegister(&PINA) & 0x0F) );
    }

    while(weiterlesen){
        lesen(drv.getRegister(&PINA) & 0x0F);
    }

    return 0;
}

void lesen(uint8_t gelesenesPaket){

    if (gelesenePakete1.size() != 0){
        if((gelesenePakete1.at(gelesenePakete1.size()-1)) != gelesenesPaket){
            gelesenePakete1.push_back(gelesenesPaket);
        }
```

```
    }

    else if(gelesenePakete1.size() == 0 ){
        if(gelesenesPaket == (startBit>>4)){
            gelesenePakete1.push_back(gelesenesPaket);
        }
    }

    if( (gelesenePakete1.size() > 3 ) && (gelesenesPaket == (endBit>>4))
){
        gelesenArrayAbfragen();
    }
}

void gelesenArrayAbfragen(){

    switch(gelesenePakete1.size()) {
        case 4:
            lesePaketeZusammensetzen(gelesenePakete1.at(1),
gelesenePakete1.at(2));
            break;

        case 6:
            if(gelesenePakete1.at(1)==readEscBit1 &&
gelesenePakete1.at(3)==readEscBit1){
                lesePaketeZusammensetzen(sendEscBit1, sendEscBit2);
                break;
            }
            else if(gelesenePakete1.at(1)==readEscBit1 &&
gelesenePakete1.at(2)==readEscBit2){
                lesePaketeZusammensetzen(startBit, gelesenePakete1.at(4));
                break;
            }
            else if(gelesenePakete1.at(2)==readEscBit1 &&
gelesenePakete1.at(3)==readEscBit2){
                lesePaketeZusammensetzen(gelesenePakete1.at(1),
gelesenePakete1.at(4));
                break;
            }
            else if(gelesenePakete1.at(3)==readEscBit1 &&
gelesenePakete1.at(4)==readEscBit2){
                lesePaketeZusammensetzen(gelesenePakete1.at(1), endBit);
                break;
            }
            break;

        case 8:

            if(gelesenePakete1.at(1)!=readEscBit1){
                lesePaketeZusammensetzen(endBit, endBit);
```

```
            }
            else if(gelesenePakete1.at(1)==readEscBit1){
                lesePaketeZusammensetzen(startBit, startBit);
            }
            break;

        case 9:
            if(weiterlesen){
                if(gelesenePakete1 == empfangeEndsignal){
                    weiterlesen = false;
                    prüfsummeNext = true;
                    break;
                }
                else{
                    break;
                }
            }

        default:
            break;
    }
}



void charInPaketeAufteilen(uint8_t erstesPaket, uint8_t zweitesPaket){

    sendenPakete.clear();

    sendenPakete.push_back(startBit);

    if(erstesPaket==sendEscBit1 && zweitesPaket==sendEscBit2){
        sendenPakete.push_back(erstesPaket);
        sendenPakete.push_back(zweitesPaket);

        sendenPakete.push_back(erstesPaket);
        sendenPakete.push_back(zweitesPaket);

        sendenPakete.push_back(endBit);
    }
    else{
        if(startBit==erstesPaket){
            sendenPakete.push_back(sendEscBit1);

            sendenPakete.push_back(sendEscBit2);
        }

        sendenPakete.push_back(erstesPaket);

        if(erstesPaket==zweitesPaket){
```

```
            sendenPakete.push_back(sendEscBit1);

            sendenPakete.push_back(sendEscBit2);
        }

        sendenPakete.push_back(zweitesPaket);

        if(zweitesPaket==endBit){

            sendenPakete.push_back(sendEscBit1);

            sendenPakete.push_back(sendEscBit2);
        }

        sendenPakete.push_back(endBit);
    }
}


void lesePaketeZusammensetzen(uint8_t erstesPaket, uint8_t zweitesPaket){

    gelesenesZeichen = ((erstesPaket<<4) | zweitesPaket); //erstes Paket
hochshiften
    alleEmpfangeneChars.push_back(gelesenesZeichen);

    if(prüfsummeNext == false){

        std::cout << gelesenesZeichen << std::endl;
        resetLeseVariablen(); //LeseVariablen leeren
    }
    else if(prüfsummeNext){
        empfangeneDateiEmpfangenesCRC8 = ((erstesPaket<<4) | zweitesPaket);
        empfangeneDateiEigenesCRC8 = berechneCRC8(alleEmpfangeneChars);

        if (empfangeneDateiEigenesCRC8 != empfangeneDateiEmpfangenesCRC8)
        {
            std::cerr<<"Fehler in der Übertragung"<<std::endl;
        }
    }
}

void resetLeseVariablen(){
    gelesenePakete1.clear();
}


char reverse(char b){
    b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
    b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
    return b;
```

```
}

uint8_t berechneCRC8(const std::vector<uint8_t>& data) {
    const uint8_t polynomial = 0x07;  // CRC-8 polynomial (x^8 + x^2 + x^1 +
x^0)
    uint8_t crc = 0x00;

    for (uint8_t byte : data) {
        crc ^= byte;
        for (uint8_t bit = 0; bit < 8; ++bit) {
            if (crc & 0x80) {
                crc = (crc << 1) ^ polynomial;
            } else {
                crc <<= 1;
            }
        }
    }

    return crc;
}
```